fahd.blog

Let the code do the talking...

TUESDAY, AUGUST 14, 2012

Analysing a Java Core Dump

In this post, I will show you how you can debug a Java core file to see what caused your JVM to crash. I will be using a core file I generated in my previous post: Generating a Java Core Dump.

There are different ways you can diagnose a JVM crash, listed below:

The hs_err_pid log file

When a fatal error occurs in the JVM, it produces an error log file called `hs_err_pidXXXX.log`, normally in the working directory of the process or in the temporary directory for the operating system. The top of this file contains the cause of the crash and the "problematic frame". For example, mine shows:

```
$ head hs_err_pid21178.log
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  SIGSEGV (0xb) at pc=0x0000002b1d00075c, pid=21178, tid=1076017504
#
# JRE version: 6.0_21-b06
# Java VM: Java HotSpot(TM) 64-Bit Server VM (17.0-b16 mixed mode linux-amd64 )
# Problematic frame:
# C  [libnativelib.so+0x75c]  bar+0x10
#
```

There is also a stack trace:

```
Stack: [0x000000004012b000,0x000000004022c000],  sp=0x000000004022aac0,  free space=3fe0000000000000018k
Native frames: (J=compiled Java code, j=interpreted, Vv=VM code, C=native code)
C  [libnativelib.so+0x75c]  bar+0x10
C  [libnativelib.so+0x772]  foo+0xe
C  [libnativelib.so+0x78e]  Java_CoreDumper_core+0x1a
j  CoreDumper.core()V+0
j  CoreDumper.main([Ljava/lang/String;)V+7
v  ~StubRoutines::call_stub
V  [libjvm.so+0x3e756d]
```

The stack trace shows that my java method, `CoreDumper.core()`, called into JNI and died when the `bar` function was called in native code.

Debugging a Java Core Dump

In some cases, the JVM may not produce a `hs_err_pid` file, for example, if the native code abruptly aborts by calling the `abort` function. In such cases, we need to analyse the core file produced. On my machine, the operating system writes out core files to `/var/tmp/cores`. You can use the following command to see where your system is configured to write out core files to:

```
$ cat /proc/sys/kernel/core_pattern
/var/tmp/cores/%e.%p.%u.core
$ ls /var/tmp/cores
java.21178.146385.core
```

There are a few, different ways to look at core dumps:

1. Using gdb

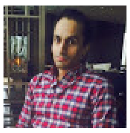
GNU Debugger (gdb) can examine a core file and work out what the program was doing when it crashed.

```
$ gdb $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
(gdb) where
#0  0x0000002a959bd26d in raise () from /lib64/tls/libc.so.6
#1  0x0000002a959bea6e in abort () from /lib64/tls/libc.so.6
#2  0x0000002b1cecf799 in bar () from libnativelib.so
#3  0x0000002b1cecf7a7 in foo () from libnativelib.so
#4  0x0000002b1cecf7c3 in Java_CoreDumper_core () from libnativelib.so
#5  0x0000002a971aac88 in ?? ()
#6  0x0000000040113800 in ?? ()
#7  0x0000002a9719fa42 in ?? ()
#8  0x000000004022ab10 in ?? ()
#9  0x0000002a9a4d5488 in ?? ()
#10 0x000000004022ab70 in ?? ()
#11 0x0000002a9a4d59c8 in ?? ()
#12 0x0000000000000000 in ?? ()
```

The `where` command prints the stack frames and shows that the `bar` function called `abort()` which caused the crash.

SUBSCRIBE TO FAHD.BLOG

113 readers BY FEEDBURNER



Fahd Shariff

Follow

View my complete profile

Hi, I'm Fahd, a software developer at an investment bank in London. I am passionate about technology and work mainly with open source software, specialising in Java applications and Unix-based operating systems.

This blog is a place for me to share useful code snippets to solve problems that I have come across, and to write about ideas and experiences as a programmer.

All code on this blog has been written by me, unless stated otherwise, and you are free to use, share and adapt it for any purpose, under the terms of the GNU General Public License.

I love hearing back from my readers, so please feel free to leave comments! Thanks for reading and happy programming :-)

Follow @fahdshariff

POPULAR POSTS

- Analysing a Java Core Dump
- Java 7: Working with Zip Files
- Changing Java Library Path at Runtime
- Retrying Operations in Java
- Increase Console Output in Eclipse [Howto]
- Useful Eclipse Templates for Faster Coding
- Spring 3 - JavaConfig: Loading a Properties File
- Writing your own Bash Completion Function
- JAXB: Marshalling/Unmarshalling Example
- Display any ResultSet in a JTable

BLOG ARCHIVE

- ► 2017 (1)
- ► 2016 (20)
- ► 2015 (13)
- ► 2014 (19)
- ► 2013 (22)
- ▼ 2012 (31)
  - ► December (3)
  - ► November (1)
  - ► October (2)

2. Using jstack

`jstack` prints stack traces of Java threads for a given core file.

```
$ jstack -J-d64 $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
Debugger attached successfully.
Server compiler detected.
JVM version is 17.0-b16
Deadlock Detection:

No deadlocks found.

Thread 16788: (state = BLOCKED)

Thread 16787: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.ref.ReferenceQueue.remove(long) @bci=44, line=118 (Interpreted frame)
 - java.lang.ref.ReferenceQueue.remove() @bci=2, line=134 (Interpreted frame)
 - java.lang.ref.Finalizer$FinalizerThread.run() @bci=3, line=159 (Interpreted frame)

Thread 16786: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.Object.wait() @bci=2, line=485 (Interpreted frame)
 - java.lang.ref.Reference$ReferenceHandler.run() @bci=46, line=116 (Interpreted frame)

Thread 16780: (state = IN_NATIVE)
 - CoreDumper.core() @bci=0 (Interpreted frame)
 - CoreDumper.main(java.lang.String[]) @bci=7, line=12 (Interpreted frame)
```

3. Using jmap

`jmap` examines a core file and prints out shared object memory maps or heap memory details.

```
$ jmap -J-d64 $JAVA_HOME/bin/java /var/tmp/cores/java.14015.146385.core
Debugger attached successfully.
Server compiler detected.
JVM version is 17.0-b16
0x0000000040000000      49K     /usr/sunjdk/1.6.0_21/bin/java
0x0000002a9566c000      124K    /lib64/tls/libpthread.so.0
0x0000002a95782000      47K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/jli/libjli.so
0x0000002a9588c000      16K     /lib64/libdl.so.2
0x0000002a95998f000     1593K   /lib64/tls/libc.so.6
0x0000002a95556000      110K    /lib64/ld-linux-x86-64.so.2
0x0000002a95bca000      11443K  /usr/sunjdk/1.6.0_21/jre/lib/amd64/server/libjvm.so
0x0000002a96699000      625K    /lib64/tls/libm.so.6
0x0000002a9681f000      56K     /lib64/tls/librt.so.1
0x0000002a96939000      65K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/libverify.so
0x0000002a96a48000      228K    /usr/sunjdk/1.6.0_21/jre/lib/amd64/libjava.so
0x0000002a96b9e000      109K    /lib64/libnsl.so.1
0x0000002a96cb6000      54K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/native_threads/libhpi.so
0x0000002a96de8000      57K     /lib64/libnss_files.so.2
0x0000002a96ef4000      551K    /lib64/libnss_db.so.2
0x0000002a97086000      89K     /usr/sunjdk/1.6.0_21/jre/lib/amd64/libzip.so
0x0000002b1cecf000      6K      /home/sharfah/tmp/jni/libnativelib.so
```

Useful Links:

Crash course on JVM crash analysis
Generating a Java Core Dump

Posted by Fahd Shariff at 11:51 AM
Labels: core-dump, gdb, Java, programming

G+1 +2 Recommend this on Google

9 comments:

sarabjeet 12:31 PM

This comment has been removed by a blog administrator.

Reply

Tutorsindia 9:48 AM

Great article..I like it..

best dissertation services

Reply

~SS~ 1:44 PM

How to perform on 32 bit windows machine

Reply

kiran m 5:43 AM

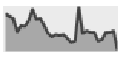
► September (7)
▼ August (5)
Java 7: Fork/Join Framework Example
Analysing a Java Core Dump
Generating a Java Core Dump
Running a command on multiple hosts
bash error: value too great for base

► June (1)
► May (1)
► April (3)
► March (3)
► February (1)
► January (4)

► 2011 (58)
► 2010 (47)
► 2009 (38)
► 2008 (44)
► 2007 (21)
► 2006 (22)

Java Code Geek

TOTAL PAGEVIEWS



1 0 3 1 3 9 5

Thanks for Sharing this valuble information and itis useful for me and CORE JAVA learners.We also provides the best Online CORE JAVA Training classes.

Reply

任喜军 3:21 AM

Great idea, I don't know the jstack can attach to a core dump file. After read this article I try many times like : jstack -J-d64 $JAVA_HOME/bin/java /var/tmp/hs_err_pid20216.log(I get this file when the jvm crash), always fail:

```
jstack -J-d64 $JAVA_HOME/bin/java /old/Downloads/hs_err_pid20216.log
Attaching to core /old/Downloads/hs_err_pid20216.log from executable /usr/local/jdk1.6.0_32//bin/java, please wait...
Error attaching to core file: Can't attach to the core file
```

Reply

Hand Green 2:18 PM

Is there a way to analyze the mdmp file (http://docs.oracle.com/cd/E15289_01/doc.40/e15059/crash.htm)?

Reply

wikiconsole 6:56 AM

This comment has been removed by the author.

Reply

wikiconsole 6:56 AM

Very Useful article.

I found following article useful too:

http://www.wikiconsole.com/wiki/introduction-to-heap-dump-core-dump/

Reply

SAP Online Trainings 5:34 PM

This comment has been removed by the author.

Reply

Enter your comment...

Comment as: Select profile... ▼

Publish Preview

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)